10 November 1998

Standard Applications Framework for EPICS (SAFE)

Steve Lewis (LBNL)

Revision 2.0—for EPICS R3.13Beta12-LBL.2 or R3.13.0-LBL.2

1.0 Introduction

The Standard Applications Framework for EPICS (SAFE) provides an organized approach for building EPICS applications on a Unix host, include the cross-compiled VxWorks parts, the native Unix parts, and ancillary tools. SAFE provides disk directory layouts, scripts, recipes, and documents. A particular feature of SAFE is the separation of the VxWorks/booting and IOC/initialization processes into two phases, each of which takes place in its own *area*, called *IOC* and *Application*, respectively. Each area has a creation script that builds and populates its directories, based on the SAFE configuration file and those from EPICS.

This document describes the procedures necessary for any project to use SAFE to develop and operate IOC applications. Although SAFE proper resides in the Application and IOC areas, its correct functioning is dependent upon some conventions on disk layout, on user/group naming, on network configuration, on installing VxWorks, and on installing EPICS.

Although initially much installation activity takes place in the files associated with VxWorks and EPICS, except for updates these files remain quiescent. The Application and IOC areas refer to them with symbolic links. Most other development activity takes place in the IOC and Application areas, and all **operational** activity (actually “driving” the system) takes place in the Application area.

Although this document will cover some aspects of the VxWorks and EPICS installations, such information is intended only to support understanding of creating and using the Application and IOC areas; full installation of VxWorks and EPICS is assumed to be provided by an experienced person.

This version refers to EPICS release 3.13beta12 or 3.13.0. It will support VxWorks version 5.3.1 for the 680x0 and PPC families.

2.0 General

2.1 Directory Layout

A minimum of 500MB on a dedicated partition is recommended. Unless otherwise noted, the same group is used for all directories (such as “epics”) and group-write permission is enabled, so root access is not needed for shared maintenance. Consider then such a partition:

```
/project/                       “$PROJECT”
      master/
      devel/
      repository/               “$CVSROOT”
```

where the three main subdirectories are for master copies, development, and the CVS repository. It is convenient to define:

```
setenv $PROJECT /project
setenv $CVSROOT /project/repository
```

The `master` subdirectory holds any external sources, such as EPICS, VxWorks, Tcl/Tk, Interviews, and so forth. These are typically not locally configured, but represent read-only originals. The convention is to install each under its generic name, with a subdirectory for the version, for example:

```
/project/master/
            vxworks/
                  vxw_5.2/
                  vxw_5.3.1/
                  .../
            epics/
            tcltk/
            iv/
            .../
```

The development area has the same generic- and version-level structure, except a symbolic link `release` points to the current release of each package. It also contains the “boot” area which is described fully in the IOC Area and Application Area sections.

```
/project/devel/
            vxworks/
            epics/
                  R3.12.2-LBL.6/
                  R3.13.0-LBL.1/
                  ...
                  release@
            boot/
```

Two methods are used to flesh out the development tree: *shadowing* and CVS import/checkout. I use shadowing for VxWorks, since very little is modified: In this method, a symbolic link is made from the version-level directory in `devel` to its counterpart in `master`; then the `realize` script is repeatedly invoked to “drill down” to the few files actually needed for local configuration (RCS can be utilized on a per-file basis). I use CVS for EPICS, since more files are modified. Each method is further described in later sections.

2.2 Unix Administrative Issues

All IOCs will boot with a unique identity. Initially they use `rsh` to load the kernel, then NFS for making file accesses after proper mounts are performed. (I do not recommend booting with `ftp` nor making file accesses with `rsh`.) In addition to each developer and operator having an account, a special account is created just for the IOCs. In the example following, the names and numbers are, of course, just illustrative; follow normal conventions for your site.

- Create an entry in `/etc/passwd`:

  ```
  vxuser:*:1000:100:VxWorks:/project/devel/boot:/bin/csh
  ```

- At least two groups are needed: One for VxWorks installation + maintenance/use; and another for EPICS installation + maintenance/use. (A more elaborate scheme would have four separate groups.) Although the IOC account is by default in the VxWorks group, it is also in the EPICS/use group and uses that for NFS activity. Create entries in `/etc/group`:

  ```
  vxgroup:*:100:maintainer,...
  ```
  (people qualified for VxWorks maintenance)

  and

  ```
  epics:*:101:vxuser,maintainer,...
  ```
  (people qualified for EPICS maintenance and/or use)

- Create the file `.rhosts` in `/home/project/devel/boot` with user "vxuser" and group "vxgroup" and permissions `-rwxrwxr-w`. Put the IP names for all IOCs in it:

  ```
  iocA vxuser
  ...
  ```

- The IOC IP address/name pairs should also be entered in `/etc/hosts`.
- Enable NFS export for `/home/project`:

  SunOS, HPUX (`/etc/exports`):

  ```
  /project -access=iocA:...
  ```

  Solaris (`/etc/dfs/dfstab`):

  ```
  share -F nfs -o rw=iocA:... /project
  ```

  where `iocA:...` is the expanded list of IOC names; or a "netgroup" mechanism may be used.

3.0 Installing VxWorks

Install from the CD-ROM into `master/vxworks/<release>`. (You will need the installation keys, but not the Tornado license.) Owner should be you or root, group should be "vxgroup", generally everything read-only. Then, in `devel/vxworks`, create the symbolic link `../../master/vxworks/<release>` and using `realize`, work down the hierarchy ("`realize <dir>; cd <dir>`"), finally creating only two actual files plus one more for each BSP (where `<bsp>` = `mv177`, …, using standard VxWorks notation):

```
/project/devel/vxworks/<release>/
                    target/config/
                          all/
                                configAll.h
                                usrConfig.c
                          <bsp>/
```

```
            sysLib.c
      <bsp>/
            sysLib.c
      .../
```

At this point I recommend changing the default VxWorks configuration away from the WRS default. It will serve as a template for the local copies made later by `iocMakeDir` (see IOC Area section) or can even be used directly. For standard EPICS, within `configAll.h` the following "facilities" should be enabled by moving the `#define` statements from the "excluded" to the "included" section:

```
INCLUDE_CONFIGURATION_5_2     /* pre-tornado tools */
INCLUDE_NFS                   /* nfs package */
INCLUDE_PING                  /* outbound ping utility */
INCLUDE_RDB                   /* remote debugging (old) */
INCLUDE_SECURITY              /* shell security */
INCLUDE_POSIX_ALL             /* all POSIX functions */
```

You may optionally use the following for **bundled** C++:

```
INCLUDE_CPLUS                 /* C++ support */
INCLUDE_CPLUS_IOSTREAMS       /* C++ iostreams classes */
```

To enable use of windSh (which includes Tornado-style debugging) yet retain the kernel symbol table, loader, and so forth, as **required** by EPICS, use:

```
INCLUDE_SYM_TBL_SYNC          /* synchronize host
                              and target symbol tables */
```

- `INCLUDE_CONFIGURATION_5_2` defeats "Tornado" and reverts to the simple approach the EPICS community has been using for 10 years. For now, it is **required** for EPICS. It means the shell and symbol table are on the target, not on Unix. (It also means you don't need the host-based licensing scheme!) This "facility" also enables rlogin, telnet, some diagnostics, and usage of a start-up script (see EPICS section).
- `INCLUDE_NFS` is **required** by SAFE, greatly speeds things up, and enables other features. It depends on the NFS export (see Unix Administration section).
- `INCLUDE_SECURITY` is one I insist on, especially with rlogin and telnet enabled. Although the default username/password are well known to WRS customers they are not likely known to the general Internet community. I recommend that each project choose its own standard; these is done by editing the `#define`'s in `configAll.h` (vxencrypt must be run in a shell):

  ```
  LOGIN_USER_NAME  "target"
  LOGIN_PASSWORD  "bReb99RRed"  /* "password" after vxencrypt*/
  ```

- `INCLUDE_POSIX_ALL` is also required by EPICS and encourages portable code.
- `INCLUDE_SYM_TBL_SYNC` is available only in version 5.3.1 or later. It keeps the in-kernel and windSh symbol tables synchronized. (You must separately license the windSh, configure it properly, and initiate the target server with "`-s`" option.) With this option, the older RDB is automatically replaced by the newer WDB. For VxWorks 5.3 or earlier, which lacked this feature, windSh and EPICS are **incompatible**.
- `usrConfig.c` is where any processor registers need to be changed, for example, setting up Transparent Translation registers, special DMA, etc.
- `sysLib.c` is where the default mapping from VME space to processor space is kept. WRS by default maps A16, A24, and a limited amount of A32 (see Appendix C for details).

It is worth building a kernel and booting it directly, even though the full SAFE recapitulates the process locally (see IOC Area section). You must "source" the following, which you can simplify for a single OS[1]. You may quickly synthesize it from `site/vxwSetup` and `start-up/Site.cshrc` in the EPICS area if you have already performed part of that installation (see EPICS section)[2].

```
setenv WIND_BASE $PROJECT/devel/vxworks/release
if ( "$ARCH" == "solaris" ) then
      setenv WIND_HOST_TYPE sun4-solaris2
else if ( "$ARCH" == "sun4" ) then
      setenv WIND_HOST_TYPE sun4-sunos4
else if ( "$ARCH" == "hp700" ) then
      setenv WIND_HOST_TYPE parisc-hpux9
else
      setenv WIND_HOST_TYPE unknown
      echo Warning: WIND_HOST_TYPE not set!
endif


setenv PATH ${WIND_BASE}/host/${WIND_HOST_TYPE}/
                              bin:${PATH}
setenv MANPATH ${WIND_BASE}/host/man:${MANPATH}
setenv MANPATH ${WIND_BASE}/host/${WIND_HOST_TYPE}/
                              man:${MANPATH}
setenv MANPATH ${WIND_BASE}/target/man:${MANPATH
```

If the path is right, "`which make`" will find the one in `host/$WIND_HOST_TYPE/bin/make`. "`make`" should build a few .o's, some other files, and then `VxWorks` and `VxWorks.sym`. These latter two files are what is actually "booted."

You may wish to verify the VxWorks installation now, or wait until you have created an IOC area. The advantage of the latter choice is that the non-volatile RAM (NVRAM) parameters are derived for you. In any case, you should install the VxWorks boot ROMs or program the flash-RAM as appropriate for your board type.

In order to set up the NVRAM, you must connect to the console port of the VME board, either from a terminal, from a terminal server, or from the serial port of a workstation using the `tip` facility, `kermit`, or equivalent[3].

At power-on or reset you should see:

```
Press any key to stop auto-boot...
[VxWorks Boot]:
```

Try "`help`" and familiarize yourself with the print ("`p`"), change ("`c`"), and boot ("`@`") syntax. Then change the parameters to something like:

```
boot device          : ei
processor number     : 0
host name            : myhost
```

1. I always set `$ARCH` early in the login process.
2. The example is correct for VxWorks 5.3.1. Earlier versions, such as 5.2, require a different setup.
3. Sun systems typically have a reference to the "B" serial port called "hardwire."

```
file name           : /project/devel/vxWorks/
                       release/target/config/<bsp>/vxWorks
inet on ethernet (e) : 127.0.0.1:ffffff00
inet on backplane (b):
host inet (h)        : 127.0.0.2
gateway inet (g)     :
user (u)             : vxuser
ftp password (pw) (blank = use rsh):
flags (f)            : 0x0
target name (tn)     : myTarget
startup script (s)   :
other (o)            :
```

where your "host name", "host inet", and "netmask" can be found on a Sun, for example, with: "`ifconfig le0`"; and your target's "inet on ethernet" can be found from "`nslookup myTarget`". The tricky bit is that the "inet-on-ethernet" really requires the inet and netmask to appear on the one line with the "`:`" between them. The "user" should correspond to the owner of the `.rhosts` file (see Unix Administration section). Also, inspect the output of "`ifconfig le0`" to ensure that BROADCAST is enabled (for EPICS).

You should get a successful boot with a banner and then the shell prompt: "`>`". The date on the banner should match the `vxWorks` you built. `rlogin`, `telnet`, `ping`, and so forth should work. Note, typically, the absence of an entry for "gateway-inet." If the Unix host you are booting from is **not** on the same "wire." that is, is **not** connected solely by hubs and switches, you will have to enter the router's IP address there.[1]

If the "vxuser" is legitimate and you have a properly prepared the file "`.rhosts`" vxWorks will use `rsh` for accessing the host file system it booted from (`ls`, `cd`, and so forth work).

4.0 Installing EPICS

The naming scheme is "`R<APS-release>-<SITE>.<version>`", for example: "`R3.13.0-LBL.1`". (The LBL EPICS is slightly modified from a standard APS release; this document describes only the LBL type.)

The upper directory layout is:

```
/project/master/epics/
                  <version>/
                  .../
```

and

```
/project/devel/epics/
                  <version>/
```

1. As a general security precaution, note that as I have set things up, access to the VxWorks prompt gives much read and some write access to your Unix file system. I strongly recommend not **ever** adding a default route, thus defeating most access from hosts not on your "wire." If you are in a site where many offices and building share one large, bridged network, you should consider creating a small, isolated subnet with just the IOCs and selected hosts on it. In any case, if you need to allow TCP access from beyond your nearest router, use a `hostAdd` and `routeAdd` just for specific hosts. Templates for this are provided by the installation script for the IOC area.

```
.../
release@
```

where `release` points to the current release. Within each version are five main subdirectories (`site` is found only in the LBL types and in particular supports SAFE itself):

```
startup/
config/
base/
extensions/
site/
```

The installation is based on CVS. I recommend at least CVS version 1.6 and prefer 1.9. To aid in maintaining an audit trail, I add to `$CVSROOT/CVSROOT/loginfo`:

```
ALL   /usr/ucb/mail -s %s maintainer,...
```

To aid in import, checkout, and export it is convenient to add to $CVSROOT/CVSROOT/modules

```
base        epics/base
config      epics/config
extensions  epics/extensions
site        epics/site
startup     epics/startup
devel       -a base config site startup extensions
```

- Normal installation typically begins with extraction of a "tar"-format archive within the master version directory. Then, using the LBL designation for "`<version>`", for example, "`R3.13.0-LBL.1`" everything is imported into CVS:

  ```
  cd /project/master/epics
  mkdir R3.13.0-LBL.1; cd R3.13.0-LBL.1
  tar xf <tar-file>
  cvs import epics R3-13 R3-13-0-LBL-1
  ```

- The next step is to checkout a working copy, using a local designation for "`<version>`", for example, "`R3.13.0-SITE.1`" :

  ```
  cd /project/devel/epics
  cvs checkout -d R3.13.0-SITE.1 -N epics
  ```

  which creates the directory as well. Since the "extensions" to EPICS are not directly tied to a release or version, which really describes the "base," sometime it is useful to only checkout a portion of EPICS:

  ```
  cvs checkout -d R3.13.0-SITE.1 -N startup site base
  ```

- At this point you must turn aside from EPICS itself and install the following auxiliary programs:

  – Sun C/C++ unbundled compiler: `/usr/lang` on SunOS and `/opt/SUNWspro` on Solaris. If you cannot obtain the Sun compiler, it is possible to copy binary versions of all of the EPICS Unix executables from LBL to their installation directories, however, in some cases (such as Tcl/Tk), they will only run when placed in exactly the same path). Gnu C/C++ (`gcc`/`g++`) will build all of base and most of extensions. `/usr/ucb/cc` **does not work**;

  – X11R6 and Motif (bundled with Solaris); X11R5 is acceptable;

  – Tcl/Tk 7.4/4.0: I maintain sources for Tcl-7.4, Tk-4.0, Blt-1.8, and Tcl-dp3.3b1. No other combination is known to work. See Appendix A for installation recipe;

– Interviews: special “LBL for EPICS” version with headers and pre-built `lib*.a`’s. See Appendix B for installation recipe;

- Now 10 files must be tailored for your site (and four more for each additional host type); you will need to have available much of the information you have used in the previous VxWorks and EPICS installation steps. Each tailorable file is named `ro.<file>`; it should be copied to `<file>` for modification. (“ro” means “read-only; this keeps the CVS changes isolated from required site dependencies.) In particular, tailor:

  – `startup/Site.cshrc` and `Site.profile`; define host (“`HOST_ARCH`”) for Unix building; host-type (“`WIND_HOST_TYPE`”) for VxWorks building;

  – `config/CONFIG_SITE`: defines location of base and extensions;

  – `config/CONFIG_SITE.Unix.<host>`: defines X, Tcl/Tk, Motif, Interviews paths;

  – `base/config/CONFIG_SITE`: defines VME board types; compilers; location of base and extensions;

  – `base/config/CONFIG_SITE_ENV`: defines timezone; IP addresses for NFS, NNTP, LOG servers;

  – `base/config/CONFIG_SITE.Host.<host>`: defines gnu, Tcl/Tk paths;

  – `base/config/CONFIG_SITE_HOST_ARCH.<host>`: defines Perl path;

  – `site/vxwSetup`: defines VxWorks paths;

  – `site/CONFIG_SITE_SAFE`: configures SAFE (follow the included comments closely and see Configuration section);

  Note: it is neither necessary nor desirable to have multiple copies of “extensions” when there are multiple copies of base; `CONFIG_SITE` in each release does the coupling. (Of course, a symbolic link will work as well.)

  Note: “base” will contain executables for **all** hosts that share the file system (for example, SunOS and Solaris—“`sun4`” and “`solaris`”) and loadables for all targets for which you have BSPs. Do not checkout separate versions for this reason!

- “`source startup/Site.cshrc; source site/vxwSetup`”. This step should be done **once** for each shell before performing any other actions with regard to EPICS in that shell.
- “`cd base; “gmake`”. It will build the host executables and target loadables. (For later auditing, I recommend redirecting output: “`gmake >&makelog`”.) Repeat on alternate host types if appropriate; use “`gmake install.host`” since the target loadables were built in the first step. If you have copied the Unix executables because you do not have the unbundled Unix compiler, tailor `config/CONFIG_BASE` as directed by the comments within it; alternatively, use “`gmake install.cross`”.

  Note: The Unix executables in “base” are configuration tools, **not** Channel Access clients.

  I have tailored the “base” of EPICS to build a large and useful set of record, device, and driver object files; further customizing and selection for each application is handled by procedures described in the Application Area section. Note: Certain assumptions about where various VME modules appear in the VME address spaces are built-in (see Hardware Setup section).
- “`cd ../config; ./bldCfgLnks`”; makes symbolic links to “base/config” so site-dependent items do not need to be duplicated for “extensions”.
- “`cd ../extensions; gmake >&makelog`”. (Omit if you have copied the Unix executables because you do not have the unbundled Unix compiler; in fact, do not checkout the source from CVS.) As does “base”, “extensions” will contain executables for all hosts that share the file system, so repeat the procedure on all host types.

Note: Only a subset of extensions are called out; with the exception of "BURT," these are just the ones listed in my Web page "Recommended EPICS documents." You can tailor the subset in `config/CONFIG_EXTENSIONS`.

You may notice that `vxWorks` is copied into the loadables area. It is not used by SAFE.

- "`cd ../site; ./bldEnvData`"; extracts much detail from "base" and "config" and creates "`resource.def`" and "`envSet`" which will be used in the Application Area section. This is strictly for SAFE.
- "`cd ../base/bin/<host>;`
  `ln -s ../../../extensions/bin/<host>/gdct313 gdct;`
  `ln -s ../../../extensions/bin/<host>/dct313 dct`";
  places simpler references to these configuration tools (which are **not** Channel Access clients) into "base" with like tools.
- Host startup. There are two host executables intended for operating EPICS:

  - `iocLogServer`. There should be one of these running as a Unix daemon for the whole site. Typically an initialization line is added to `rc.local` or equivalent for a selected workstation (details upon request). It provides a very primitive global occurrence logger. By default, this capability is disabled in the IOCs by SAFE.

  - `caRepeater`. There should be one of these running as a Unix daemon for each workstation. Usually, the first instance of any Unix Channel-access client will cause `caRepeater` to be spawned (details upon request).

5.0 IOC Area

5.1 General

The IOC area provides the following facilities:

- Linking to particular versions of EPICS and VxWorks is mostly driven from EPICS configuration files by means of a special script "`iocMakeDir`".
- The boot lines stored in the NVRAM of the VME board never change. An initial set is derived for each IOC based on configuration files.
- Each "logical" IOC—a combination of a particular board type plus its usage—has its own area.
- Switching between applications or combining several applications on one IOC is easily done.
- The VxWorks kernel can be customized for each IOC without altering the master installation, yet site-wide configurations may be shared.
- A per-IOC writable directory is easily accessible from applications which can thus use generic file names.
- Several IOCs can run same the same application relatively easily.
- Tailoring of network parameters that are common to all IOCs is kept in one place An initial set of commands is derived based on configuration files.

5.2 Detailed Layout

```
/project/devel/boot/iocs/              "/vxboot"
      all/
            netInit.cmd
```

```
iocA/
      IOC_iocA
      log/                        "/log"
      cmd/
            example.cmd
            none.cm
            ...
      appinit.cmd@
      configVx/
            all/
                  configAll.h
                  usrConfig.c
                  ...
            <bsp>/
                  sysLib.c
                  ...
                  vxWorks
                  vxWorks.sym
            .../
      vxWorks@
      vxWorks.sym@
      NVRAM.<bsp>
      startup.cmd
      [links to EPICS and VxWorks]
iocB/
.../
```

The `all` area is shared among IOCs. `netInit.cmd` has host additions, routing additions, and NFS mounts which refer to the host or to common network parameters. The primary mounts are:

- `/vxboot` to refer to IOC area root;
- `/log` to provide generic (writable) logging area **per-IOC**;
- `/apps` to refer to Application area root (see Application Area section).

5.3 Usage

An IOC area is built by invoking from within the `iocs` root directory the script:

```
<path-to>/iocMakeDir [-v] [-LINK] <ioc> <bsp>
```

where:

- `-v` optionally requests verbose output;
- `-LINK` optionally makes symbolic links for `vxWorks` directly to the VxWorks installation rather than building a copy of the kernel locally; by default, `iocMakeDir` is in the `COPY` mode;
- `<ioc>` is the name of the IOC: `iocA`, `iocB`, `...` in the layout example;
- `<bsp>` is one of the VxWorks standard BSP designations (`mv177`, …).

`iocMakeDir` is configured by the EPICS configuration files `config/CONFIG*` and additionally by `site/CONFIG_SITE_SAFE` (see Configuration section). It is called with an **absolute path** so that it

may "lock" to a specific version of EPICS. `iocMakeDir` creates and/or fills in directories and files. Note carefully all informational, warning and error messages and take appropriate action. `iocMakeDir` never over-writes actual files, thus, it may be re-run over the same area to replace missing files or, by first removing them, to update files. (The symbolic links to EPICS are replaced—a way of switching EPICS versions. If this is your purpose, any *unmodified* files should be deleted, so you do get the latest versions, and any *modified* ones renamed for subsequent merging.)

- The first required argument is an existing or new IOC name. If the name already exists, then missing files or subdirectories are filled in; **no actual files are over-written**. A typical usage would be to change the BSP (target) type. (Changing the BSP will leave previous versions intact.)
- The second required argument specifies the BSP (target) type. The latter **must** refer to a target which you have called out in the `CROSS_COMPILER_TARGET_ARCHS` line in `config/CONFIG_SITE` and which has undergone the EPICS build process.
- If you wish to have all IOCs use identical kernels as built in the VxWorks area, specify `-LINK` as an optional argument. `LINK` defeats the ability to have variant kernel configurations for each IOC and requires that the VxWorks distribution itself be NFS mounted by `netInit.cmd` (as described below). With `LINK`, there is no local copy of `vxWorks`, just a symbolic link back to the kernel you built in the VxWorks area (`devel/vxworks/target/config/<bsp>`). Be sure that kernel is properly configured and built.
- If you want a verbose commentary as `iocMakeDir` runs, specify `-v` as an optional argument.

`iocMakeDir` builds the directory structure such as "iocA" above, and populates it with links to the EPICS area. It creates the subdirectories `cmd` and `configVx` and the files `startup.cmd`, `IOC_<ioc>` and `NVRAM.<bsp>`.

- `IOC_<ioc>` is a 'marker' used by `iocMakeDir`.
- `NVRAM.<bsp>` is a prototype of the entries to be made in the non-volatile ram (NVRAM) area of the IOC, either at the boot ROM prompt (see VxWorks section) or from a running VxWorks system, using `bootChange`. The accuracy of it is dependent on the correctness of `site/CONFIG_SITE_SAFE`. If you previously set the NVRAM up for booting directly from the VxWorks installation, you should now change the two lines "file name" and "startup script" after carefully checking the prototype.
- `startup.cmd` is the initial set of shell commands for the IOC. Again, carefully check it.

If the `-LINK` option was not used when `iocMakeDir` was invoked, `iocMakeDir` populated the `configVx` subdirectory with copies of the relevant files from the VxWorks area (`configAll.h`, `usrConfig.c`, `sysLib.c`, and others); these can be tailored for just this IOC[1]. To build the kernel (even with no tailoring):

```
source startup/Site.cshrc
source site/vxwSetup
cd configVx/<bsp>
make
```

A number of `.o` files and others will be created, then finally `vxWorks` and `vxWorks.sym`.

Changing the BSP will leave `configVx/<bsp>` directories intact. You can revert quickly to a previously built BSP by either re-invoking `iocMakeDir`, or by switching the links `vxWorks*` (and possibly `appinit.cmd`).

1. There are links to other files which, in general, should not be modified. Should you decide to do so, `realize` each file first.

The first invocation of `iocMakeDir` within the IOC area will also build a shared subdirectory `all`. The `netInit.cmd` there should be carefully checked; its accuracy is dependent on the correctness of `site/CONFIG_SITE_SAFE`.

- Be sure the paths for NFS mounts "`/apps`" and "`/vxboot`" are correct—subsequent scripts depend on them.
- The templates for two other NFS mounts are left disabled. However:

  - The "epics" NFS mount may need to be enabled if the default (LINK) option was taken with `app-Makedir`, in order that the link `target<bsp>` ~> `base/bin/<bsp>` ~> `<path-to-epics-release>/base` can be resolved from the perspective of VxWorks (see `COPY` option in Application section).

  - The "vxworks" NFS mount may need to be enabled if the `LINK` option was used in `iocMakeDir`, in order that the link `vxWorks` ~> `<path-to-vxworks-version>/target/config/<bsp>/vxWorks` can be resolved from the perspective of VxWorks.
- Templates are provided for adding hosts and routes for machines not on the local "wire."[1] (See the last part of the VxWorks Installation section for more details.)

`appinit.cmd` is a link, used by `startup.cmd`, to proceed automatically from the VxWorks primary boot process directly into loading an EPICS application. The `cmd` subdirectory contains small auxiliary command files to act as "bridges." Initially, `appinit.cmd` is linked to `none.cmd`, a null file, which ends the boot process cleanly with no EPICS application running. Also provided is a template, `example.cmd`, which should be copied and tailored for any actual application(s). Then, to move between no application or any particular application requires changing only the link `appinit.cmd`; neither the NVRAM nor the master `startup.cmd` file needs to be edited. From the null state, an application can easily be loaded manually; at the VxWorks prompt:

```
> cd "/apps/myApp"
> <cmd/startup.<bsp>.cmd
```

The uid/gid in the `nfsAuthUnixSet` line should match the "vxuser" passwd and group entries. When combined with the Unix permissions, the IOC should be able to read all the mounted areas and write into /log; make sure group-write is set on the latter and that the group is correct.

6.0 Application Area

The Application area provides the following facilities:

- Linkage to EPICS and VxWorks driven from EPICS configuration by means of special script `app-MakeDir`.
- Builds for multiple targets.
- Incremental building supported.
- Leverages heavily from EPICS Makefiles—no rules needed, just lists
- Customized record, device, and driver support provided, leaving main EPICS unaltered:

  – allows minimal loading on per-application basis;
  – easy to try new/modified types.

1. With later versions of VxWorks, if the IOC is booted through a gateway (router), the appropriate entries are made automatically. `hostShow` and `routeShow` are useful in confirming this.

Most of the “driving” activity takes place in the Application areas: configuring records, screens, alarms, archiving, *etc*, and then actively manipulating hardware from a running IOC.

6.1 Detailed Layout

```
/project/devel/boot/apps                    “/apps”
                  appX/
                        APP_appX
                        Makefile
                        db/
                        src/
                        rec/
                        dev/
                        drv/
                        dbd/
                        cmd/
                        resource.def
                        envSet
                        dl/
                        alh/                [optional]
                        ar/                 [optional]
                        target<bsp>@
                        ...
                        [links to EPICS and VxWorks]
                  appY/
                  .../
```

6.2 Usage: Create phase

An Application area is created by invoking from within the `apps` root directory the script:

```
<path-to>/appMakeDir [-v] [-COPY] <app> [<target> ...
                                          |ALL|NONE]
```

where:

- `-v` optionally requests verbose output;
- `-COPY` optionally copies generic EPICS files rather than linking to them; by default, `appMakeDir` is in the `LINK` mode;
- `<app>` is the name of the Application area (`appX`, `appY`, `...` in the layout above);
- `<target>` is an optional list of one or more VxWorks standard BSPs (`mv177`, …); or `ALL`; or `NONE`.

`appMakeDir` is configured by the EPICS configuration files `config/CONFIG*`, `base/config/CONFIG*`, and additionally by `site/CONFIG_SITE_SAFE` (see Configuration section). It is called with an **absolute** path so that it may “lock” on to a specific version of EPICS. `appMakedir` creates and/or fills in directories and files. Note carefully all informational, warning and error messages and take appropriate action.

- The `COPY` option replaces links with copies of the generic EPICS files `iocCore`, `seq`, `initHookLib`, and `devLibOpt`. Each symbolic link `target<bsp>` becomes a subdirectory into which the files are placed. This option is useful if the EPICS master area will **not** be accessible to VxWorks

by an NFS mount using that symbolic link during EPICS loading (for example, if EPICS is in an AFS or DFS area). Note carefully that the default `LINK` option, may require manually configuring an NFS mount in the `all/netInit.cmd` file in the IOC area (see IOC area section).

- The first required argument is an existing or new Application name. If the name already exists, then missing files or subdirectories are filled in; **no actual files are over-written**. A typical usage would be to add another BSP. However, the symbolic links to EPICS are replaced—a way of switching EPICS versions. (If this is your purpose, any *unmodified* files should be deleted, and any *modified* ones renamed for subsequent merging; and typically then you would use the "`NONE`" argument for the target type.)
- The second required argument specifies one or more BSP (target) types. Each of these **must** refer to a target which you have called out in EPICS in the `CROSS_COMPILER_TARGET_ARCHS` line in `config/CONFIG_SITE`. If "`ALL`" is specified, the entire list in this line is used. If the argument is `NONE` the BSP lists and files are left intact—this is useful when upgrading an existing Application area to a new version of EPICS, or when replacing missing or damaged files with originals.

The fundamental application is developed in the `db` and `src` subdirectories:

- `db` is where the configuration of the "function-block" database is prepared. This is the list of "records" and their non-default parameters that comprise the heart of an EPICS application. A variety of techniques are available. If `gdct` ("Graphical Database Configuration Tool) or `dct` are used, they will key off the defining `<app>.dbd` file; a link is pointed to the actual file (see Build instructions, below). As explained in the Configuration section, provision is also made for supporting CAPFAST.
- When needed, "C" code for the functional parts of subroutine records, and SNL code for sequencers, is prepared in the `src` subdirectory. A template Makefile is provided; follow the comments to specify your actual source and object files; the built-in EPICS rules are sufficient for typical usage. Any standalone VxWorks tasks may easily be accommodated here, as well as any Unix programs.

The `rec`, `dev`, `drv` and `dbd` subdirectories are populated with files copied and slightly modified from within the master EPICS area. These four together perform two functions: "tailoring" each Application area to use just those existing records and devices/drivers needed; and "customizing" each Application by modifying or adding new records and devices/drivers, without affecting the master EPICS.

Initially, `rec`, `dev`, and `drv` contain only `recBaseLIBOBJS`, `devBaseLIBOBJS`, and `drvBaseLIBOBJS`, respectively. These are modified from the EPICS originals by disabling all "hardware" types, leaving only "software" types. The loading of EPICS requires, ultimately, three additional components in addition to the four in the `target<bsp>` group: `recLib`, `devLib`, and `drvLib`. These are built-up by pre-linking the individual VxWorks object modules from the master EPICS area locally. To add standard hardware types already installed in EPICS, the appropriate lines in the respective `{rec,dev,drv}BaseLIBOBJS` are enabled (by removing the comment symbol and whitespace). To add custom types, the modified or new source code is prepared locally (for example, `xxx.c`), and one of the following three methods used:

1) add to `Makefile.Vx` the lines

```
LIBOBJS += xxx.o ...
RECTYPES += xxx.h ... [rec only]
```

2) add to `Makefile.Vx` the line

```
include xxxBaseLIBOBJS
```

and create `xxxBaseLIBOBJS` with the lines

```
LIBOBJS += xxx.o ...
RECTYPES += xxx.h ... [rec only]
```

...

3) add directly to {rec, or dev, or drv}BaseLIBOBJS the lines

LIBOBJS += xxx.o ...

RECTYPES += xxx.h ... [rec only]

(Internal comments mark the appropriate places.) The modified or new object module will appear only locally. Using the built-in EPICS RULES, gmake will perform the actions (see Build phase, below). Note that this action must be perfectly synchronized with modification of the appropriate dbd/*.dbd file.

- dbd provides database definition files that are put through a series of special "build" processes, producing a master file called <app>.dbd. This serves two purposes: it guides the initialization of EPICS in VxWorks so just those records, devices, and drivers called out are initialized, using the specified default field values, by dbLoadDatabase(); and it constrains the database construction tools under Unix (gdct and dct) to allow only the desired entities, to prompt with only allowed options, and to produce only the non-default fields used by the dbLoadRecords() and dbLoadTemplate() procedures used to complete the initialization of EPICS.

  There are five files, initially copied and modified, in dbd:

  - baseRec.dbd: calls out records to be included in recLib. **Note** that any changes must be perfectly synchronized to changes in rec/Makefile.Vx and/or rec/recBaseLIBOBJS.

  - baseDev.dbd: calls out devices to be included in devLib. **Note** that any changes must be perfectly synchronized to changes in dev/Makefile.Vx and/or dev/devBaseLIBOBJS.

  - baseDrv.dbd: calls out drivers to be included in drvLib. **Note** that any changes must be perfectly synchronized to changes in drv/Makefile.Vx and/or drv/drvBaseLIBOBJS. (Special note: it is not possible to specify no drivers at all, so drvDummy exists to make things work; if one of more real drivers are called out, it may be disabled.) Where possible, drivers have been bundled with their devices, so many hardware types appear only in baseDev.dbd.

  - baseBpt.dbd: calls out breakpoint tables to be included in devLib. For customization, follow the prescription in Chapter 2 of "IOC Application Developer's Guide".

  - baseMenu.dbd: calls out menu enumerations to be included in devLib. For customization, follow the prescription in Chapter 2 of "IOC Application Developer's Guide".

  To add standard hardware types already installed in EPICS, the appropriate lines in the respective {rec, or dev, or drv}.dbd are enabled (by removing the comment symbol and any whitespace). To add custom types, the modified or new database definition is prepared locally (for example, xxx.dbd), and one of the following three methods used:

  1) add to each of base{rec, or dev, or drv}.dbd the respective lines

  record ... or

  device ... or

  driver ...

  2) add to xxxInclude.dbd the lines

  include recxxx.dbd or

  include devxxx.dbd or

  include drvxxx.dbd

  and create xxx.dbd with the respective lines

  record ... or

`device` ... or

`driver` ...

3) add directly to `<app>Include.dbd` the lines

`record` ... or

`device` ... or

`driver` ...

(Internal comments mark the appropriate places.) `<app>Include.dbd` serves as a master organizing file for gmake, allowing (with the `Makefile.Host`)all of the above possibilities. There is considerable flexibility, and the various inclusions may be mixed in any order that makes maintenance easier.

- `cmd` subdirectory holds the master startup file called from the IOC area; it calls five auxiliary files in turn.
- `resource.def` and `envSet`: these are directly constructed from the configuration files of EPICS, and are used, respectively, for initializing the pseudo-environment variables of EPICS in VxWorks; and for initializing the shell environment when "driving" the Application area (see Build phase, below).
- `dl` (and optionally `alh`, `ar`) subdirectories: hold the configuration files specific to the `edd`/`dm` (and optionally `alh`, `ar`) tools. Some initial templates are provided:

  - in `dl`, a link to a suggested `Color.adl` containing a small, documented set of colors, and a few useful sets of color dynamics rules;

  - in `ar`, a subdirectory `arReq` containing three sample request files and an initialization file `open.dat`; plus a logging directory `arLog`;

  - in `alh`, a sample configuration file.

6.3 Usage: Build phase

Before any further activity, and when subsequently restarting activity, in **each** such shell, do **once**:

```
source site/epicsSetup
```

This will pick up the local environment variables, and will also conditionally "source" the VxWorks setup file (`site/vxwSetup`) and the EPICS setup file (`startup/Site.cshrc`) in the proper order.

> **It is important to understand that this initialization activity is *context-dependent,* that is, it *must* be done from within the *specific* Application Area you are going to use. It refers to the symbolic links which bind this Application Area to a particular version of EPICS + VxWorks. It sets the customized, synchronized "environments" defined by `envSet` and `resource.def` for the Unix-client and VxWorks-server, respectively, that define this application. It can also be done from within the "site" directory of an EPICS release; this would set up for a generic or default Application. The process will *not* work from within your "home" `.cshrc` or `.login`, since 1) the symbolic links are not there; and 2) even if you put in absolute paths or duplicate links you are binding to just one application, EPICS, and VxWorks for all shell windows.**

- `Site.cshrc` sets the `HOST_ARCH` and `WIND_HOST_TYPE` environment variables; thus it is needed by both build and run phases of `appMakeDir`, which will attempt to call it if you have not done so.

- `vxwSetup` sets the `WIND_BASE` environment variable and puts the VxWorks tools and manual pages in your path.
- `epicsSetup` sets the environment variables of envSet; and it puts the base tools (`gdct`, `snc`, *etc*), extensions (`edd`, `dm`, `alh`, `ar`, *etc*) and EPICS manual pages in your path. It does this so everything is accessible from within the main Application directory and any first-level subdirectory. This is convenient for invoking `gdct` in `dbl`, `edd` or `dm` in `dl`, *etc*; or any of them from the main directory. `epicsSetup` calls `Site.cshrc` and `vxwSetup` if you have not already done so, thus you may simple use it as a one-step setup.

To actually build or rebuild the tailored or customized EPICS components and any "C" or SNC code you have written, you may either invoke (with no arguments): "`gmake`" from the top level (recommended once immediately after the Create phase); or individually, `gmake` to incorporate any changes in `rec`, `drv`, `dev`, `dbd`, or `src` from within the respective subdirectories. Note that "`gmake`" will build for the host type you invoke it from and all target types[1].

- Before loading your application into the IOC, you must modify some of the application-specific files in `cmd`:

  - `epicsCode.<bsp>.cmd` should **not** be modified;

  - `userCode.<bsp>.cmd` to load your subroutine record procedures, sequencers, and any stand-alone tasks from `bin/<bsp>`;[2]

  - `db.cmd` to load and expand your `db/*.db` and `*.tmpl` files using `dbLoadRecords()` and `dbLoadTemplate()` as needed;

  - `epicsInit.cmd` to perform EPICS housekeeping and to select EPICS features such as logging, hot-start, and time-synchronization;

  - `userInit.cmd` to initiate your sequencers and/or to start any stand-alone tasks.

  A master coordination file `startup.<bsp>.cmd` calls the above in order; it should **not** be modified for a single application. This is the file called by `appinit.cmd` from the IOC area to proceed from the VxWorks boot to the EPICS initialization. Note that this file is BSP-dependent.

  For multiple applications, you must carefully interleave the several instances of `db.cmd`, `userCode.<bsp>.cmd`, and `userInit.cmd` with **one** invocation of `epicsCode.<bsp>.cmd` and `epicsInit.cmd`; building a combining master coordination file in a separate area is one method.

6.4 Usage: Run phase

When actually "driving" the Application, you should be in its top directory. Specific OPI tools, such as `edd`/`dm`, `alh`, *etc*, typically have sufficient path definitions or symbolic links such that they will run from there *or* from their unique subdirectory.

- `dm`, in particular, honors the environment variable `EPICS_DISPLAY_PATH`. This ":" delimited list specifies directories to search for screens. To allow sharing of some screens across Applications, append additional dl directories to it. The default setting allows for finding screens in the local application:

```
.:./dl:../dl
```

1. You may constrain `gmake` by giving it the argument `<goal>.<arch>`, where `<goal>` is either `install` or `build` and `<arch>` is a Unix host or VxWorks BSP (such as `solaris` or `mv177`) or a generic class (such as `host` or `cross`). Example: `gmake build.mv177` or `gmake install.host`.
2. Every `<bsp>` type must be so modified.

To find more screens, say, in the Network Statistics Application, change it (in `envSet`) to:

```
.:./dl:../dl:../stats/dl:../../stats/dl
```

- When the IOCs and OPIs ("consoles") are not all on the same subnetwork, then the OPI tools and IOC channel access servers cannot "find" each other. The method of finding is controlled by EPICS_CA_ADDR_LIST. You will have to modify the environment for both categories as suggested below.

  To enable an IOC to reach OPIs on other subnets, add their IP addresses to `EPICS_CA_ADDR_LIST` in `resource.def`. You must also add a `hostAdd()` and a `routeAdd()` to the `startup.cmd` sequence, typically in `cmd/epicsInit.cmd` for an isolated IOC or in `all/netInit.cmd` for all IOCs. (An exception to the latter requirement is that if an IOC is booted with a gateway, the host/route entries are automatically entered for the gateway.) If a router that serves a distant subnet allows it, then you can enable access to **all** the OPIs on that subnet by using the subnet address in `EPICS_CA_ADDR_LIST` and just the routeAdd().

  Do **NOT** use the `defaultRoute()` feature, as it presents a serious security breach.

  Assume IOC 111.222.33.44 wants to reach OPI 111.222.3.4 and the router on the IOC subnet is 111.222.33.1. Then either:

  ```
  EPICS_CA_ADDR_LIST DBF_STRING 111.222.3.4
  routeAdd "111.222.3.4","111.222.33.1"
  hostAdd "remoteOPIName","111.222.3.4"
  ```

  or

  ```
  EPICS_CA_ADDR_LIST DBF_STRING 111.22.3.0
  routeAdd "111.222.3.4","111.222.33.1"
  ```

  To enable an OPI to reach IOCs on other subnets, add their IP addresses to `EPICS_CA_ADDR_LIST` in `envSet`. If a router that serves a distant subnet allows it (a feature called "directed broadcast"), then you can enable access to **all** the IOCs on that subnet by using the subnet address in `EPICS_CA_ADDR_LIST`.

  Assume OPI 111.222.3.4 wants to reach IOC 111.222.33.44. Then either:

  ```
  setenv EPICS_CA_ADDR_LIST 111.222.33.44
  ```

  or

  ```
  setenv EPICS_CA_ADDR_LIST 111.222.33.0
  ```

7.0 Configuration

The file `CONFIG_SITE_SAFE` determines many parameters used by the scripts `iocMakeDir` and by `appMakeDir` when building new areas; these scripts also use parameters contained in the standard EPICS `config` and `base/config` directories. Some of the parameters are only advisory, for example, those used in constructing the NVRAM file which suggests how to setup the VME board. The parameters are:

7.1 Common

- EPICS_ROOT: the root of the EPICS version to which the area will be linked. It must be a directory containing at least the subdirectories: `startup`, `site`, `config`; it is required whenever `iocMakeDir` and `appMakeDir` are not called with an absolute path.

7.2 For `iocMakeDir`

- SAFE_BOOT_HOST is the Unix host for booting IOCs; defaults to current host. It will be used for "host name" in NVRAM and for NFS mounts in startup scripts. Needed only if you work from other, NFS-mounted hosts.
- SAFE_BOOT_IP is the IP address for SAFE_BOOT_HOST. If not supplied, then `nslookup` is used; if that fails, EPICS_TS_NTP_INET or EPICS_IOC_LOG_INET from EPICS "config" will be used. A value of "`127.0.0.1`" indicates all of the above failed.
- SAFE_BOOT_TGT_IP is the IP for the VME target board (IOC); **NOT to be used if you have multiple targets**. If not supplied, then `nslookup` on <target> is used. A value of "`127.0.0.1`" indicates all of the above failed.
- SAFE_BOOT_NETMASK is the "netmask" for the local LAN of the target. If not supplied, then `ifconfig` is used (see "LANIF" section below). A value of "`ffffff00`" is used if all of the above fail. If `iocMakedir` will be run from hosts with different LAN interfaces, or from a different LAN from where the targets will boot, this parameter *should* be supplied.
- SAFE_BOOT_PATH is the absolute path to the `boot` area; if not supplied, the directory above the current working directory is used. A warning is issued if the final resolved path used does not contain the subdirectories `apps` and `iocs`.
- SAFE_PROJ is the absolute path to the EPICS and VxWorks area; if not supplied "`/project`" is the default. It is used to construct suggested NFS mounts for the `all/netInit.cmd` file.
- SAFE_VX_USER, SAFE_VX_USER_UID, SAFE_VX_GROUP, SAFE_VX_USER_GID are the "user" and "uid" in `/etc/passwd`, and "group" and "gid" in `/etc/group` as target will appear to host. (See Unix Administrative Issues section for details about this.)

7.3 For `appMakeDir`

- AUX_DIRS is a list of additional directories used for EPICS tools or other purposes, for example, `ar` and `alh`; templates may exist to fill them out.
- CAPFAST: "`true`" if you want subdirectories filled out within `db` for typical use of this tool; "`false`" if you are using GDCT or another method.

7.4 LANIF and BOOT_DEV setup

- For each VxWorks BSP (target) type, a typical LAN interface boot device abbreviation is listed under "Specific Boot Devices." If your BSP is not shown, consult documentation or try "help" from the VxWorks boot prompt and see "available boot devices" to add to this list.
- For each host type, a typical LAN interface type is listed. If your host type is listed, confirm manually that the LANIF entry works; change it if required, or add an entry if you have a new host type. For the case where you have several hosts with different LANIF types, this method is fails and the netmask should be supplied (see Configuration section)

8.0 Booting

Here is the dynamic thread of the booting sequence.

- `startup.cmd` will be invoked after the kernel is loaded. It will:
  - tailor the console and login prompts;
  - call `netInit.cmd` for the common commands;
  - mount `/log` for all applications
  - call `appinit.cmd` to load the application(s)
  - ends.
- `netInit.cmd:`
  - performs the NFS mounts and other common commands;
  - returns.
- `appinit.cmd`
  - changes to the application subdirectory;
  - invokes the local, target-specific `startup.<bsp>.cmd;`
  - returns.
- `startup.<bsp>.cmd:`
  - invokes the local, target-specific `epicsCode.<bsp>.cmd;`
  - invokes the local, target-specific `userCode.<bsp>.cmd;`
  - invokes the local `db.cmd;`
  - invokes the local `epicsInit.cmd;`
  - invokes the local `userInit.cmd;`
  - returns.

In order to load multiple applications into one IOC, the generic application startup's must be broken apart (see Application area section).

9.0 Appendix A: Installing Tcl/Tk

Please consult `http://csg.lbl.gov/csg.html` for a current pointer. Obtain the four `gzip`'ed "`tar`"-format files for tcl, tk, blt and dp. Create a directory tree as:

```
/home/project/master/tcltk/src/
```

move into it, and `gunzip` and extract the "`tar`"-format files, resulting in:

```
tcl7.4/
tk4.0/
blt-1.8
tcl-dp3.3b1
```

- Building tcl: Adjust your `PATH` to ensure that the unbundled C compiler is found before `/usr/ucb/bin/cc`.

```
cd tcl7.4
./configure --prefix /home/project/master/tcltk
make; make install
```

- Building tk:

```
cd tk4.0
./configure --prefix /home/project/master/tcltk
make; make install
cd ../../lib
ln -s libtk4.0.a libtk.a
```

- Building blt:

```
cd ../src/blt-1.8
./configure --prefix /home/project/master/tcltk
```

Edit `Makefile`, changing the definition of `TK_LIB` from `-ltk4.0` to `-ltk`; and changing the definition of `TCL_LIB` from `-ltcl7.4` to `-ltcl`.

```
make; make install
```

- Building dp:

```
cd ../src/tcl-dp3.3b1
```

Edit `configure.users`, changing the definition of:

`DP_ROOT` to `/home/project/master/tcltk`
`TCL_INCLUDE` to `../tcl7.4`
`TCL_LIBRARY` to `../tcl7.4/library`
`TCL_LIBDIR` to `../tcl7.4`
`TK_INCLUDE` to `../tk4.0`
`TK_LIBRARY` to `../tk4.0/library`
`TK_LIBDIR` to `../tk4.0`

Prepend to your `PATH` `/home/project/master/tcltk/bin`.

```
./configure --with-cc
make; make install
```

Now the directory structure should look like:

```
/home/project/master/tcltk/
                          src/
                                tcl7.4/
                                tk4.0/
                                blt-1.8
                                tcl-dp3.3b1
                          bin
                                tclsh
                                ...
                          lib
                                libtcl.a
                                ...
                          include
                                tcl.h
                                ...
                          man
```

10.0 Appendix B: Installing Interviews

Please consult `http://csg.lbl.gov/csg.html` for a current pointer. Obtain the gzip'ed tar-format "`include`" file and the two "`lib.a`" files for each flavor of Unix. Create a directory tree as:

```
/home/project/master/iv/
                        lib/
                              O.<arch>
                              ...
```

where `<arch>` follows the EPICS convention for host architecture. Move into the top, and `gunzip` and extract the "`tar`"-format "`include`" file. Then move into each `lib/O.<arch>`, `gunzip` the appropriate two "`lib.a`" files, and rename them, resulting in:

```
lib/
      O.<arch>/
            libIV.a
            libUnidraw.a
      ...
include/
      Interviews/
      Unidraw/
      ...
```

11.0 Appendix C: VME Hardware and Driver Notes

One of the most difficult aspects of EPICS is assigning addresses to modules plugged into the VME backplane. The current scheme, which applies to all EPICS version through 3.12, leaves much to be desired. (A great improvement comes with version 3.13.)

11.1 `Module_types.h`

The information is contained in `base/include/module_types.h`. All current drivers have been assigned a position in an array; the entries describe the start address, number of modules ("slots"), interrupt level, and so forth. This is a common module, loosely maintained by the EPICS community. This piece of code requires a great deal of study before attempting modification **:-)**.

By way of introduction, the major scheme is as follows: the are four major sections, for "ai", "ao", "bi", and "bo" types, respectively. Within each section, there are arrays describing the known board types. The position in the array is called out by a `#define` which will be recognized by the driver/device code. The arrays must be carefully filled out, respecting the position of the entry you are concerned with. The most important arrays are:

`num_cards`: the number of cards for which "slots" are expected;
`num_channels`: the number of channels allowed per card;
`interruptible`: 0/1 denoting whether the device does not/does support interrupts;
`bus`: the formal EPICS bus enumeration (VME, CAMAC, A-B, *etc*);
`addrs`: the address in the VME short-I/O (A16) space;
`memaddrs`: the address in the VME standard (A24) or extended (A32) space.

Note that the latter two types are mutually exclusive.

For each VME board type, the address switches are set to one of the "slots" that are pre-defined for it; the drivers probe the VME address space during initialization. Boards within a type do not need to be in contiguous slots.

The advantage of this method is that no special initialization calls are made during IOC startup: if a record refers to a device, and the driver is configured in, it will find the hardware. Another advantage is that once non-overlapping addresses have been built-in for all known board types, there will never be an address conflict within a specific VME crate. Each site will have its own version, and these definitions may be over-ridden by adding additional code in `module_types.c` or even with VxWorks command-shell lines.

11.2 VME vs. processor addresses

The addresses found in `module_types.h` are true VME addresses, as set in the switches and jumpers of the I/O boards. From within the code running under VxWorks, however, *processor* addresses are used. The relationship between these is set by hardware (such as the VMEchip2 ASIC of the MVME-167/177 series) which connects the CPU board's internal bus to the VME bus, and by software, particularly in a structure in `sysLib.c` called `sysPhysMemDesc`. For each IOC, you can modify this structure to arrange just the amount of processor address you want assigned to the A32 ("extended"), A24 ("standard") and A15 ("short I/O") spaces of the VME bus.

In general, you should not change the A16 and A24 mappings at all. In a typical Motorola VME board, A24 has the full 16 MBytes mapped from `0xf000'0000` to `0xf0ff'ffff`, and A16 has the full 64KBytes mapped from `0xffff'0000` to `0xffff'ffff`. Also, a small, 16Mbytes portion of A32 is mapped from `0x0200'0000` to `0x02ff'ffff`. You can map much greater amounts than this (up to nearly 2 Gbytes), but doing so will consume large amounts of real RAM and take a long time to initialize.

You may also change the caching. The VME space should always have caching disabled. By default, WRS sets D32 accesses to A24 devices to be changed to two sequential D16 accesses; for some VME boards, this is just the wrong tactic and must be modified.

You should follow the convention that all EPICS drivers do, and **never** use absolute processor addresses; use the VME addresses and the function `sysBusToLocalAddrs`() to translate to processor address. The demonstration application "vmeProbe" is an excellent example of this; you are encouraged to use it for actual hardware testing and as a template for coding. (Notice careful use of the function `vmeMemProbe`() as well.) In general, I do not recommend attempting to access the VME bus directly from the VxWorks shell; often you will get a false, negative result merely because you have failed to properly translate to processor space or have not put enough "casts" in to generate the proper data width instructions (D8, D16, D32).

12.0 Example Applications

A "tar"-format file is provided for each; use `appMakeDir` to create a new Application area, then extract the file (from within the new Application directory); follow the steps in the HOW-TO file; and build using `appMakedir` again.

12.1 VMEbus Probe

This application has a new record type, "`vmeProbe`." It is a mechanism for probing the VME address space to aid in integrating new VME boards, verifying switch settings, bit patterns, *etc*.

The capability is provided to read or write from the A16, A24, or A32 address spaces using D8, D16, or D32 transfers, either single-word or multiple-word. Both value and status are returned, thus, there is no side-effect from probing into unimplemented address ranges.

In `db`, the new record is instantiated, along with a number of auxiliary "soft" records to act is input and output buffers.

In `dl`, a screen was built for actual use.

12.2 Simulator

This application has two layers: the lower layer uses the *simulate* feature of EPICS records to bypass actual hardware, allowing a simple CALC record to mimic a power-supply with a slow response time. The upper layer uses a sequencer to control the simulated supply. Two independent supplies are instantiated.

The application also demonstrates the hot-start feature of the IOC: All dynamic parameters are restored after a reboot. Notice the reference to "bumpless" parameters in epicsInit.cmd; the saved data appears in the /log area of the IOC in use.

A master screen is provided, which can invoke both layers of the independent simulations.

12.3 Network Statistics

This Application adds a network statistics monitor capability to a system of one or more IOCs. For each IOC, the memory, cpu-usage, and Channel-Access client activity is monitored.

After building the STATS application itself, instructions are included to add the hooks to each primary application in each IOC. In the STATS dl subdirectory there is a master network mimic diagram which should be customized for your system. The generic screens do not need change. All the screens are accessible from the primary application areas.

For unusually heavy or light application loads, you may want to tailor the thresholds to provide the correct degree on sensitivity; this is done by changing the "warning" and "alarm" values in db/`ioc.tmpl` by using `db/stats.db` as a guide. (Hint: read the `.DESC` fields.)